Novel "Squiral" (square spiral) architecture for fast image processing

Novel "Squiral" (Square Spiral) Architecture for Fast Image Processing

Min Jing[a,*], Bryan Scotney[a], Sonya Coleman[a], T. Martin McGinnity[a,b]

[a] *Ulster University, United Kingdom*
[b] *Nottingham Trent University, United Kingdom*

Abstract

Fast image processing is a key element in achieving real-time image and video analysis. We propose a novel framework based on a square spiral (denoted as "squiral") architecture to facilitate fast image processing. Unlike conventional image pixel addressing schemes, where the pixel indices are based on two-dimensional Cartesian coordinates, the spiral addressing scheme enables the image pixel indices to be stored in a one dimensional vector, thereby accelerating the subsequent processing. We refer to the new framework as "Squiral" Image Processing (SIP). Firstly we introduce the approach for SIP conversion that transforms a standard 2D image to a 1D vector according to the proposed "squiral" architecture. Secondly we propose a non-overlapping convolution technique for SIP-based convolution, in which the SIP addressing scheme is incorporated by simulating the phenomenon of eye tremor in the human visual system. Furthermore, we develop a strategy to extend the SIP framework to be multiscale. The performance of the proposed framework is evaluated by the application of SIP-based approaches to edge and corner detection. The results demonstrate the efficiency of the proposed SIP framework compared with standard convolution.

Keywords: square spiral ("squiral") image processing (SIP), spiral addressing scheme, eye tremor, non-overlapping convolution, fast image processing

*Corresponding author
Email address: jingmin310@gmail.com (Min Jing)

1. Introduction

Real-time image processing is a challenging task, particularly when handling large-scale visual content. This work presents a novel framework for fast image processing inspired by the approach based on a hexagonal image processing (HIP) framework [1, 2, 3]. Recent work has explored the potential of the HIP framework in image processing [4, 5, 6] to speed up computation [7, 8] as HIP image pixel indices can be stored in a one-dimensional vector by incorporating a spiral architecture [9, 2]. However, the computational advantages of HIP are undermined significantly by the additional time and effort required for conversion of standard image data to a HIP environment. Because all existing hardware for capturing and displaying images is based on a rectangular pixel architecture, extra effort is required to convert a square or rectangular pixel-based image to a hexagonal pixel based structure before any indexing or processing [3]. In addition, such conversion to a hexagonal image can introduce distortions due to the hexagonal lattice not being aligned in two orthogonal directions; for example, the image may need to be shifted by a half-pixel to simulate capture on a hexagonal lattice.

In this work, we introduce a square spiral ("squiral") image processing (SIP) framework that develops a novel spiral addressing scheme for standard square pixel-based images. The advantages of SIP are fourfold. Firstly, SIP conversion can directly use the pixels in a square image which involves merely shifting points within the same Cartesian coordinate system. Secondly, SIP is designed for square images, so it can be implemented using existing hardware based on Cartesian coordinates. Thirdly, an image in SIP can be stored as a one-dimensional vector, thus retaining the computational advantage demonstrated by HIP. Fourthly, the kernel (operator) used for square images can be applied directly to SIP images, and therefore there is no need to design special operators for SIP. The advantages of SIP suggest that it can be easily incorporated with existing hardware and image processing techniques developed for rectangular images.

To implement the SIP-based framework for image processing, we developed a non-overlapping convolution technique for SIP convolution by simulating the eye tremor phenomenon of the human vision system. The characteristics of the human vision system have been explored in image processing [10, 11, 8] including several key concepts. Firstly, it is known that human eyes are never still; as proposed in [12] the "wandering of the gaze" serves to prevent retinal fatigue. Scientists today agree on the occurrence of three main types of eye movement during visual fixation in humans: tremor, drifts and microsaccades [13]. Secondly, within the central fovea of the human eye, receptive fields of ganglion cells of the same type do not overlap significantly [11]; therefore, the traditional approach to feature extraction based on overlapping convolution operators does not closely represent the human visual system. Furthermore, the human visual system does not process single static images, but instead a series of temporal images that are slightly off-set due to involuntary eye movements [10]. The use of eye tremor, rhythmic oscillations of the eye, for image processing has been exploited in [11, 8] potentially reduce the computational burden of standard convolution. A biologically-inspired HIP framework was proposed in [8] that models eye tremor using the one-dimensional addressing scheme of the spiral architecture. Inspired by [8], we develop a SIP-based non-overlapping convolution technique to facilitate fast computation on standard rectangular pixel-based images.

It is well known that the edges in images are captured in the visual context at different levels of resolution [14]. Therefore, the detection of image features in a multiscale sense is more appropriate than use of a single scale. One of the most popular ways of applying edge detection operators at multiple scales is through the use of image pyramids [15], in which the same operator (a smoothing function) is applied to images down-sampled to different scales. Alternatively, a multiscale representation of an image can be obtained by convolution of the image with a two-dimensional smoothing function defined at different scales. Studies on multiscale images have been carried out to enhance the performance of image processing [16, 17, 18, 19]. In this paper, we propose a strategy to

extend the SIP-based framework for use with multiscale operators.

The remainder of the paper is structured as follows: in Section 2 we first explain the spiral addressing scheme for rectangular pixel-based images and conversion to SIP from the standard Cartesian 2D addressing scheme. We then provide details of the development of SIP-based non-overlapping convolution involving simulation of eye tremor in Section 3, followed by the strategy to extend SIP to multiple scales in Section 4. For demonstration we use the application of edge detection and corner detection in the experiments, but the technique proposed is applicable to any convolution-based operator or feature extractor. Preliminary results are given in Section 5, which demonstrate the efficiency of the proposed SIP-based approach. Section 6 presents conclusions and future work.

2. The "Squiral" Architecture

2.1. "Squiral" Addressing Scheme

Inspired by the work in [8] we propose a new "squiral" addressing scheme for square images as shown in Fig. 1. In the rest of the paper we refer to the image based on the "squiral" architecture as the SIP image. It can be seen from

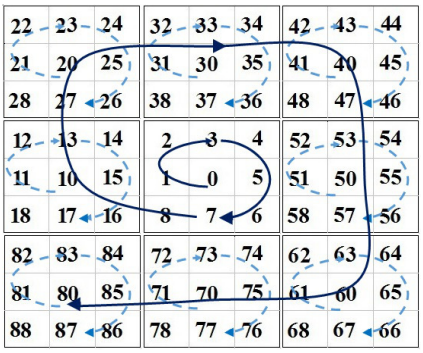


Figure 1: The *squiral* addressing scheme for a layer-2 SIP based on a square image.

Fig. 1 that the addresses start from the centre pixel (layer-0), then move from pixel 1 to 8 indexed in a clockwise direction. The cluster of the centre pixel

together with its eight (layer-0) neighbours is considered as layer-1. Layer-2 is generated by recursive use of layer-1 clusters, such that eight layer-1 clusters are combined to form layer-2. Ultimately, the entire SIP image can be considered as a layer-λ cluster comprising 9^λ pixels. This structure facilitates the use of base 9 numbering to address each pixel within the image using one-dimensional indexing. For example, the pixels in layer-1 are labelled from 0 to 8, indexed in a clockwise direction. The base 9 indexing continues into each layer, e.g. layer-2 starts from 10, 11, 12, ..., and finishes at 88. Subsequent layers are structured recursively. The converted SIP image is stored in a one-dimensional vector according to the SIP addresses. In Section 2.2 we explain in detail how to convert a square image to a SIP image.

2.2. SIP Conversion

Unlike HIP conversion, which relies on use of a re-sampling scheme from a rectangular to hexagonal image [1, 3], SIP is based on square images, and therefore no re-sampling scheme is needed. We need only to convert the lattice of a square image to the new SIP format based on the spiral addressing scheme. The details of SIP conversion are explained in the following.

2.2.1. Determine SIP Image Size

The size of a SIP image is determined by the number of SIP layers λ. For a given rectangular image with size $M \times N$, the number of SIP layers λ can be obtained by $\lambda = (\log M + \log N)/\log 9$, and the length of the SIP image (a vector) is 9^λ. For example, a small square image of size 81×81 can be converted to a layer-4 SIP image, 243×243 to layer-5 and 729×729 to layer-6, respectively. When the image is not square, we can either trim the rectangular image into a square, centred at the centre of the original image, or extend it to a square by padding with zeros. In this work, we make a square image by trimming down the rectangular image.

2.2.2. SIP Addressing

The SIP-based address of a pixel can be defined as: $a_n a_{n-1}...a_1$, where $0 \leq a_i < 9$ (because the SIP address scheme is base 9). To facilitate a straightforward indexing system, the address notation follows that used in previous HIP based work [2], in which the spiral address is represented as in the decimal system. We note that, the base 10 representation is used to facilitate the definition of spiral addition and multiplication (see below) and hence is used to denote the address, but that the address itself is a value in base 9 that corresponds to an index position (which may be counted in base 10) in a vector. For example, Fig. 1 shows the addresses in a SIP layer-2, in which the address is indexed from 0 to 88, and the total number of pixels for SIP is $9^2 = 81$. The SIP address can be represented as:

$$a_n a_{n-1}...a_1 = \sum_{i=1}^{n} a_i \times 10^{i-1} \tag{1}$$

where $\sum$ and $\times$ denote Spiral Addition and Spiral Multiplication, respectively. Spiral Addition corresponds to translation of the image and Spiral Multiplication corresponds to rotation and scaling of the image. The arithmetic of SIP operations for addition and multiplication are defined in Table 1 and Table 2, respectively. In both tables, the inputs are shown in the first column and first row, and the result of the operation on any pair of inputs is given by the value shown at the intersection of the row and column corresponding to the inputs. We note that the tables are symmetric about the diagonal: both Spiral Addition and Spiral Multiplication are commutative.

To demonstrate the use of Spiral Addition and Spiral Multiplication, we give examples below. According to Eq. (1), a SIP address 57 can be presented as:

$$57 = 5 \times 10^1 + 7 \tag{2}$$

The arithmetic operations can be illustrated as in Fig. 2, where $\oplus$ and $\otimes$ denote the arithmetic operations of Spiral Addition and Spiral Multiplication, respectively. To perform $5 \otimes 0$ based on Table 2, the input 5 from the first

Table 1: Spiral Addition Table based on SIP Addresses

$\sum$	0	1	2	3	4	5	6	7	8
0	0	1	2	3	4	5	6	7	8
1	1	15	14	2	3	0	7	8	16
2	2	14	26	38	37	3	0	1	15
3	3	2	38	37	36	4	5	0	1
4	4	3	37	36	48	52	51	5	0
5	5	0	3	4	52	51	58	6	7
6	6	7	0	5	51	58	62	74	73
7	7	8	1	0	5	6	74	73	72
8	8	16	15	1	0	7	73	72	84

Table 2: Spiral Multiplication Table based on SIP Addresses

$\times$	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8
2	0	2	3	4	5	6	7	8	1
3	0	3	4	5	6	7	8	1	2
4	0	4	5	6	7	8	1	2	3
5	0	5	6	7	8	1	2	3	4
6	0	6	7	8	1	2	3	4	5
7	0	7	8	1	2	3	4	5	6
8	0	8	1	2	3	4	5	6	7

column multiplies the input 0 specified in the first row, which returns 0. Then 5 multiplies 1 to give 5. The result of multiplication is then added to 7 based on the Spiral Addition in Table 1, i.e. $0 \bigoplus 7 = 7$, which gives 57.

Note that the Spiral Addition requires a *carry rule*. To illustrate this we give another example. Based on Table 1, the sum of 57 and 8 can be computed as: first 7 and 8 added to obtain 72. The 7 is then carried and added to 5 to produce 6. The result is thus 62. There are no carries in Spiral Multiplication (Table 2) because the effect of multiplication is only to rotate in the clockwise direction.

2.2.3. Locating the SIP Address in the 2D Image

Conversion to a SIP image from a square image can be considered as a process to locate each SIP address as a point in the 2D image represented by

$$\begin{array}{r} 10 \\ \otimes\ \ 5 \\ \hline 50 \\ \oplus\ \ 7 \\ \hline 57 \end{array}$$

Figure 2: Example of using spiral addition and spiral multiplication to obtain SIP based address.

the Cartesian coordinates (x, y). Consider layer-1 as an example, we can define the location of a centre point as $L(0) = (0, 0)$. Based on the SIP addressing scheme shown in Fig. 1, we have the location of nine points in layer-1, ordered clock-wise: $L(1) = (-1, 0), L(2) = (-1, 1), L(3) = (0, 1), L(4) = (1, 1), L(5) = (1, 0), L(6) = (1, -1), L(7) = (0, -1)$ and $L(8) = (-1, -1)$. The location can be represented as:

$$L = \left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \end{bmatrix} \right\} \quad (3)$$

To locate the points in a higher layer, we can calculate the shift required from the centre to the target point by

$$L(a_i \times 10^{i-1}) = 3^{i-1} * L(a_i) \quad (4)$$

where $*$ denotes normal multiplication. For example, based on Eq. (1) and Eq. (4), the point at SIP address $L(87)$ can be located by finding the addresses of $L(80)$ and $L(7)$, such as $L(87) = L(80) + L(7) = 3 * L(8) + L(7) = (-3, -4)$. Hence, the point $L(87)$ can be found by shifting the start point from $(0, 0)$ to $(-3, -4)$. The point $L(4536)$ can be located by $L(4536) = L(4000) + L(500) + L(30) + L(6) = 3^3 * L(4) + 3^2 * L(5) + 3 * L(3) + L(6) = (37, 29)$. Similary, the point $L(4536)$ can be found by shifting the start point from $(0, 0)$ to $(37, 29)$. Based on this process, we can convert a square image to a SIP image by locating in the original square image the pixel position whose value should be entered for each position in the one-dimensional SIP vector. A visual example of a layer-2

SIP image conversion based on the proposed SIP conversion is given in Fig. 3.

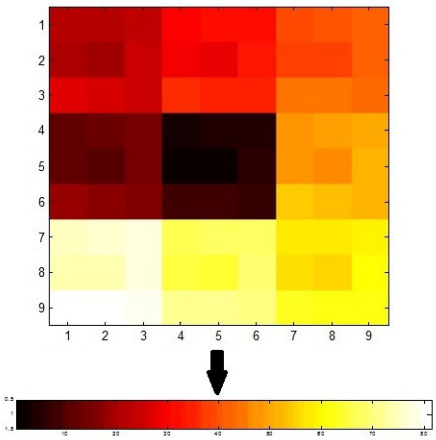


Figure 3: Illustration of the recursive spiral nature of the conversion of a square image to a SIP image obtained by the proposed method.

3. "Squiral" Image Processing (SIP) Framework

The SIP framework is illustrated in Fig. 4, which gives an example of SIP based edge detection, although the technique proposed is applicable to any convolution-based operator or feature extractor. To simplify the discussion, we use a 3×3 kernel, which can be considered as a layer-1 SIP image. The process includes: (1) mapping the original image to the SIP addressing scheme and converting the square image to a 1D SIP image; (2) applying the layer-1 "eye tremor" model (details are provided in Section 3.1), in which nine SIP images are generated by shifting the image centre; (3) non-overlapping convolution, i.e, by rearranging the SIP vector such that the kernel moves only to the centre of each sub-layer cluster (details are provided in Section 3.2); (4) SIP convolution executed by matrix-vector multiplication; (5) for display purposes, construction of a square edge map from the SIP edge detector outcomes (although this step is not always necessary as it is useful only when visual output is required).

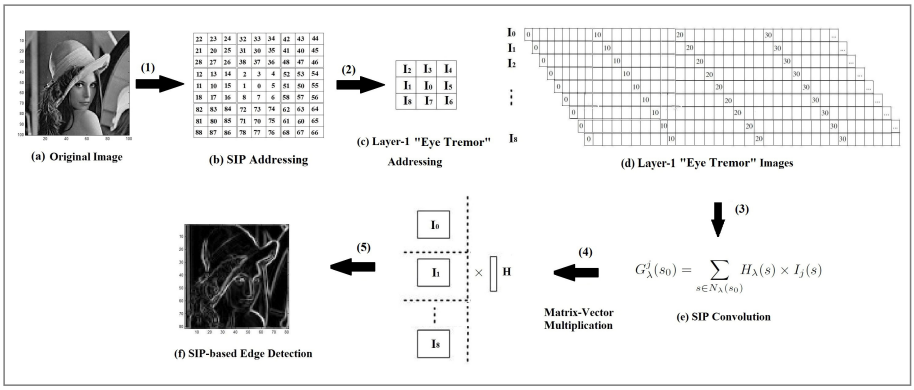


Figure 4: The proposed SIP framework for edge detection based on a layer-1 operator (3×3), which includes:(1) SIP conversion; (2) simulation of eye tremor; (3,4) SIP convolution, and (5) construction of a 2D edge map (only if presentation of outcomes in 2D is required).

3.1. Simulation of Eye Tremor

When using a standard 2D addressing scheme, the addresses of a pixel's neighbours can be determined easily. However, determining a pixel's neighbours in a 1D addressing scheme is not straightforward and requires significant computation. For HIP, determining a pixel's neighbours requires time consuming special hexagonal and radix-7 addition [3]. To tackle this issue, in [8] a method is proposed in which the concept of eye tremor is introduced. Following [8], we introduce eye tremor to the proposed SIP framework. To simplify the case, we use a 3×3 kernel as an example. The idea is illustrated in Fig. 5, which shows an example of layer-1 eye tremor involving nine SIP images, each addressed using the spiral architecture. Fig. 5 gives the addresses of the *centres* of nine eye tremor images and each of the nine SIP images is a 1D vector with the centre off-set to a neighbouring pixel. In Fig. 5, consider I_0 as a "base" SIP image, and eight additional images $I_j, j = 1, 2, ..., 8$ can be obtained by shifting I_0 by one pixel in the image plane along the proposed spiral addressing scheme. The "centre" of each image I_j is located at a pixel within the layer-1 neighbourhood centred at image I_0 as in Fig. 5. Each image is stored as a vector after being converted from the 2D image structure, which enables us to

achieve fast and efficient processing for feature extraction. Next we explain how the eye tremor SIP images are used in SIP convolution.

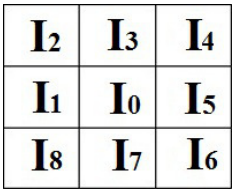


Figure 5: Pixel offsets for a layer-1 eye tremor using 9 images.

3.2. SIP Non-Overlapping Convolution

For a given image, convolution of a SIP operator (denoted as H_λ, where λ denotes the SIP operator layer) across the entire image plane is achieved by applying the operator sparsely to each of the 9^λ eye tremor images I_j, and then combining the resultant outputs. Unlike traditional 2D image convolution in which the input is a 2D image, the input of SIP convolution is the eye tremor images. For example, SIP convolution using a layer-1 operator needs nine eye tremor images, and a layer-2 operator needs 81 eye tremor images. To facilitate fast processing, the concept of non-overlapping convolution is deployed. Based on the eye tremor model, for each image I_j, we apply the operator H_λ only when centred at those pixels with spiral address $0(mod 9^\lambda)$, hence achieving non-overlapping convolution. The non-overlapping concept is illustrated in Fig. 6, which shows that a SIP operator H_1 (3×3) is applied only to the centre with spiral address $0(mod 9)$. The regions over which the SIP operator is applied are shown in different colours.

The SIP convolution of the image I_j with an operator H_λ can be represented as,

$$G^j_\lambda(s_0) = \sum_{s \in N_\lambda(s_0)} H_\lambda(s) I_j(s) \tag{5}$$

where $\forall s_0 \in \{s | s = 0(mod 9^\lambda))\}$, $N_\lambda(s_0)$ denotes the λ-neighbourhood centred on the pixel with spiral address s_0 in eye tremor image I_j, where j is the number of eye tremor images. If we take a layer-1 operator H_1 as an example (note H_1

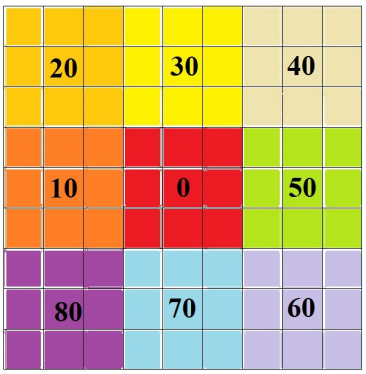


Figure 6: Illustration of SIP-based convolution showing how the operator H_1 is applied when centred with spiral address $0(mod 9)$

is equivalent to an operator size 3×3), the matrix implementation of Eq. (5) can be written as:

$$\begin{pmatrix} G_1^0 \\ G_1^1 \\ \vdots \\ G_1^8 \end{pmatrix} = \begin{pmatrix} I_0 \\ I_1 \\ \vdots \\ I_8 \end{pmatrix} \begin{pmatrix} H_1(0) \\ H_1(1) \\ \vdots \\ H_1(8) \end{pmatrix} \tag{6}$$

where the input $I_j, j = 0, 1, ..., 8$ is the nine eye tremor images, and H_1 is the layer-1 SIP operator (also a vector). The output is

$$G_1^j = \begin{pmatrix} G_1^j(0) \\ G_1^j(10) \\ \vdots \\ G_1^j(k) \end{pmatrix} \tag{7}$$

where $k = 0, 10, 20, 30, ..$ denotes the centre of each sub-cluster as shown in Fig. 6 and the eye tremor image, and

$$I_j = \begin{pmatrix} I_j(0) & I_j(1) & ... & I_j(8) \\ I_j(10) & I_j(11) & ... & I_j(18) \\ \vdots & \vdots & \ddots & \vdots \\ I_j(k) & I_j(k+1) & ... & I_j(k+8) \end{pmatrix} \tag{8}$$

The SIP convolution outcome from Eq. (6) can be obtained by assembling the values of G_1^j into a vector with the following arrangement:

$$\begin{pmatrix} G_1^0(0) & G_1^1(0) & \cdots & G_1^8(0) \\ G_1^0(10) & G_1^1(10) & \cdots & G_1^8(10) \\ \vdots & \vdots & \ddots & \vdots \\ G_1^0(k) & G_1^1(k) & \cdots & G_1^8(k) \end{pmatrix} \tag{9}$$

The final outcome in a SIP format is obtained by rearranging each row of the matrix in Eq. (9) into a vector. The process of assembling the one-dimensional vectors for SIP-based convolution is illustrated in Fig. 7.

$$[G_1^0(0)G_1^1(0)...G_1^8(0)G_1^0(10)G_1^1(10)...G_1^8(10)...G_1^0(k)...G_1^8(k)] \tag{10}$$

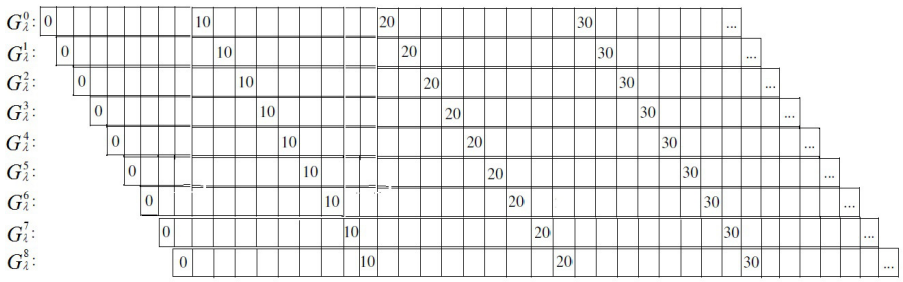


Figure 7: Illustration of assembling the one-dimensional vectors for SIP based convolution.

To demonstrate the process of SIP convolution, we first look at an example of standard 2D convolution as shown in Fig. 8, which includes a 2D matrix as input, a 3×3 kernel and the output of 2D convolution. The value of a given pixel in the output is the weighted sum obtained by multiplying each kernel value with the corresponding input image pixel values. To perform SIP-based convolution, the kernel is flipped and converted to a SIP based operator (vector). The nine eye tremor images are then generated as explained in Section 3.1. The

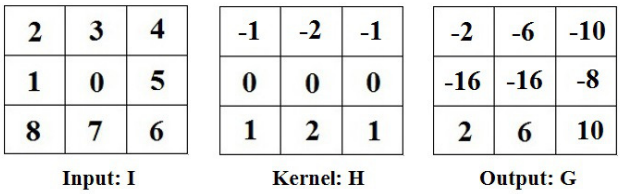


Figure 8: An example of convolution of a two 2D matrix with 3×3 kernel and the output of convolution.

SIP convolution can be executed based on Eq. (6), such as:

$$\begin{pmatrix} G(0) \\ G(1) \\ \vdots \\ G(8) \end{pmatrix} = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 0 & 0 & 2 & 3 & 0 & 7 & 8 & 0 \\ 2 & 0 & 0 & 0 & 0 & 3 & 0 & 1 & 0 \\ 3 & 2 & 0 & 0 & 0 & 4 & 5 & 0 & 1 \\ 4 & 3 & 0 & 0 & 0 & 0 & 0 & 5 & 0 \\ 5 & 0 & 3 & 4 & 0 & 0 & 0 & 6 & 7 \\ 6 & 7 & 0 & 5 & 0 & 0 & 0 & 0 & 0 \\ 7 & 8 & 1 & 0 & 5 & 6 & 0 & 0 & 0 \\ 8 & 0 & 0 & 1 & 0 & 7 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 1 \\ 0 \\ -1 \\ -2 \\ -1 \end{pmatrix} = \begin{pmatrix} -16 \\ -16 \\ -2 \\ -6 \\ -10 \\ -8 \\ 10 \\ 6 \\ 2 \end{pmatrix} \tag{11}$$

If required for presentational purposes, the SIP convolution results can be restructured to a 2D output, as illustrated in Fig. 8. As SIP and standard convolution differ only in the approaches used for pixel indexing and image storage, the results of convolution are the same for both methods.

3.3. SIP at Multiple Scales

To obtain a multiscale representation of an image, a popular approach is the use of image pyramids [15], in which the same operator is applied to down sampled images at different scales. Here we present an alternative approach of applying the operator at different scales. Fig. 9 shows the addressing for the elements of multiscale SIP operators. It can be seen that an operator of size 3×3 can be converted into a layer-1 SIP vector. For SIP-based convolution Eq. (5), the length of the SIP operator H_λ matches the required number of

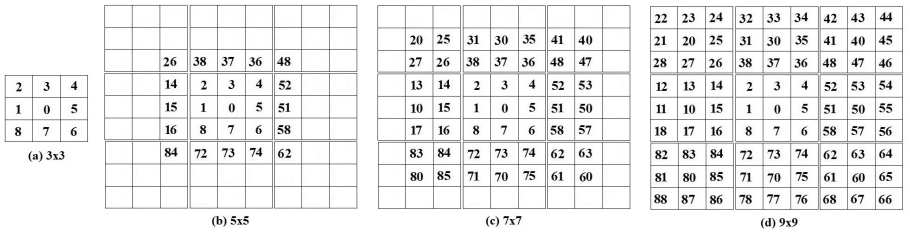


Figure 9: Index addresses for multiscale SIP operators: (a) 3×3; (b) 5×5; (c) 7×7 and (d) 9×9.

eye tremor images. Since this number increases by a factor of 9 as we move up between layer sizes, intermediately sized operators may require padding by zeros. For example, operators of size 5×5 and 7×7 are converted to a layer-2 SIP vector by padding with zeros to extend them to size 9×9. By this process SIP convolution may be applied with operators at multiple scales.

4. Experimental Results

In the experiments we used three well-known test images: Lena, peppers and coins, with image sizes of 100×100, 384×520 and 738×900, respectively. The centre of each image was located first. The image was then trimmed down and converted into a SIP image, in which the number of SIP layers for each are: layer-4 (81×81), layer-5 (243×243) and layer-6 (729×729), respectively. For SIP conversion, we pre-calculated the coordinate shifting table up to layer-6 (total 533,441 pixels). This look-up table needs to be calculated only once and then saved for use in any SIP conversion. We evaluate the performance of SIP-based convolution using edge detection and corner detection as two application examples and compare the results to standard convolution in terms of the feature detection maps and execution run times. Implementation of the The feature detection was performed for the same size of image (in terms of numbers of pixels) as the SIP images. Because the focus of the experiments is on SIP convolution, the time required for SIP conversion and construction of eye tremor images were not considered for the comparative evaluations. For fair compari-

son, to analyse algorithms independently of specific implementations, hardware, or data, all algorithms were coded using Matlab and executed based on a 64-bit Windows operating system, Intel processor core i7. Undoubtedly faster performance would be obtained using other programming languages. SIP framework based on C and parallel implementation are currently under development, and it is planned to report the results of that work in the future.

4.1. SIP Convolution using a 3×3 *Kernel*

In this experiment we compare the SIP approach to both the standard 2D convolution and Matlab built-in function *conv2.m*. Implementation of standard 2D convolution involves use of four *for* loops that move the flipped kernel along each row and column of the input image and compute a weighted sum over the neighbourhood. The Matlab function *conv2.m* is optimised by separable convolution, in which the 2D convolution is performed by two 1D convolutions (assuming that the 2D filter is separable), so is much faster than the standard 2D convolution approach. To compare the run times, three kernel functions (size 3×3) were used: Sobel, Gaussian (separable) and Laplacian (unseparable).

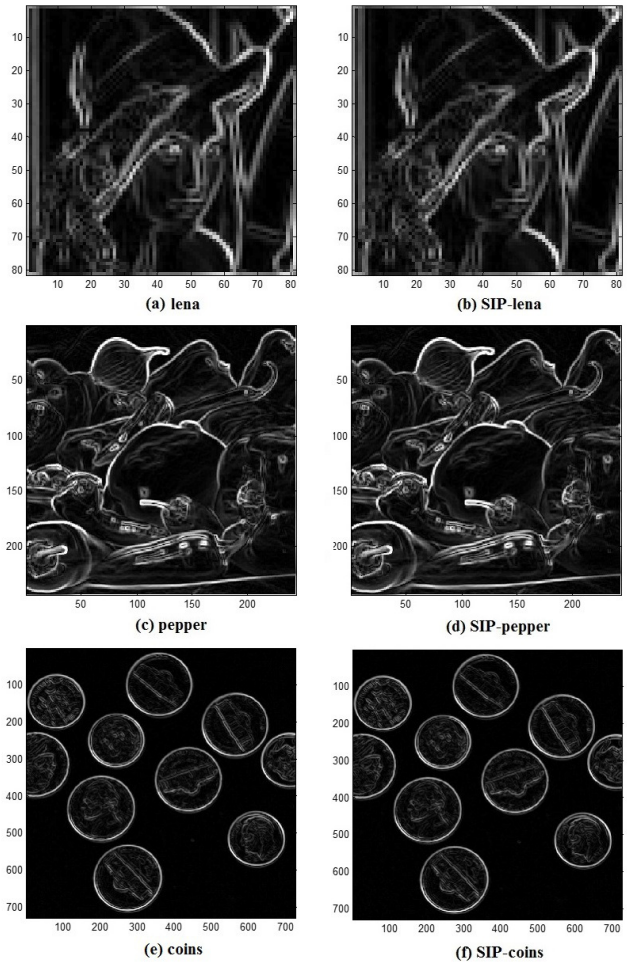


Figure 10: Comparison of edge results from standard 2D convolution and SIP-based convolution for layer-4 (Lena), layer-5 (pepper) and layer-6 (coins) SIP images.

The results of edge detection using the Sobel operator are shown in Fig. 10. The left-hand column shows the results from standard 2D convolution and the right-hand column shows the results from the SIP approach. As SIP and standard convolution differ only in the approaches used for pixel indexing and image storage, the results of convolution are the same for both methods. Correlation of edge results between the SIP and standard methods are all equal to 1.0. The run-times based on an average of 100 runs for the Sobel, Gaussian and

Laplacian operators are presented in Table 3, Table 4 and Table 5, respectively. It can be seen that the SIP approach is not only much faster than standard 2D convolution, but also comparable to the Matlab *conv2.m* function for both separable and unseparable operators.

Statistical significance tests were carried out for the three methods based on the run time results over the images of layer-4, layer-5 and layer-6. The p values obtained from t-tests are given in Table 6, in which $p < 0.05$ indicates statistically significant difference in performance. The results show that for each image size, there is significant difference in performance between standard convolution and SIP, and that SIP is comparable in performance to the optimised Matlab.

Table 3: Average Run-times for Sobel (Seconds)

	Layer-4	Layer-5	layer-6
Standard	0.006826	0.061737	0.605872
SIP	0.000282	0.001130	0.016618
Matlab	0.000284	0.001305	0.014501

Table 4: Average Run-times for the Gaussian Operator (Seconds)

	Layer-4	Layer-5	layer-6
Standard	0.0047	0.0468	0.4597
SIP	0.00006054	0.0009565	0.0125
Matlab	0.00019616	0.0009039	0.0099

Table 5: Average Run-times for the Laplacian Operator(Seconds)

	Layer-4	Layer-5	layer-6
Standard	0.0026	0.024	0.2386
SIP	0.000047725	0.0004795	0.0053
Matlab	0.000045615	0.0004078	0.0033

Table 6: Comparison of p-values from T-test

	Layer-4	Layer-5	layer-6
SIP vs Standard	0.0200	0.0168	0.0166
SIP vs Matlab	0.6839	0.9609	0.6545

4.2. SIP Edge Detection at Mutltiscale Levels

To implement the SIP multiscale operators for edge detection, we used the Canny edge detector to generate four operators with size 3×3, 5×5, 7×7 and 9×9, which were then converted to SIP operators as illustrated in Fig.9. The results from multiscale SIP edge detection are shown in Fig. 11, representing the magnitude of gradient outputs.

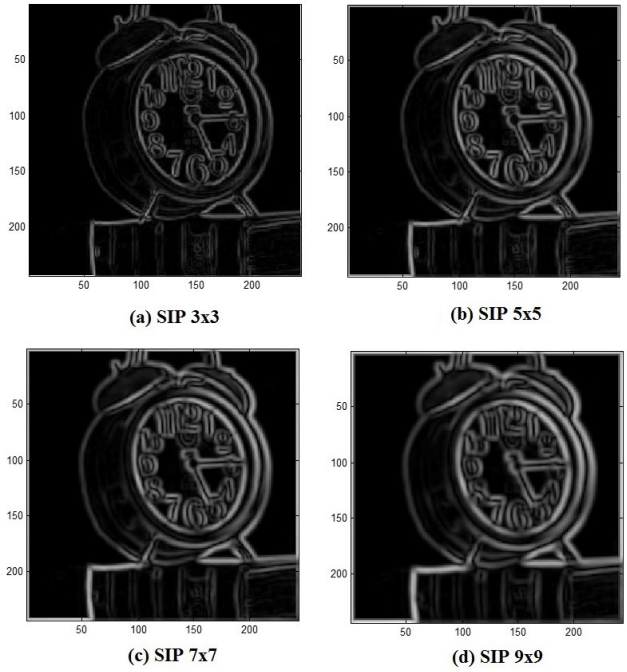


Figure 11: The results of edge detection based on SIP multiscale operators.

The results of run-times based on averages over 100 runs are given in Fig. 12 and Fig. 13, which show comparisons across all image sizes for the standard 2D convolution and SIP method, respectively. From the results we can see that for all cases, SIP is faster than standard 2D convolution. For the standard method, the computational cost increases as the sizes of images and operators increase. For the SIP-based approach, run time increases slightly with the image size, but time used for 5×5, 7×7 and 9×9 remains similar as they are all based on a layer-2 operator of size 9×9. For example, it can be seen from the y-axis that

the SIP approach for results from 7×7 and 9×9 are at least 10 times faster than those from the standard approach.

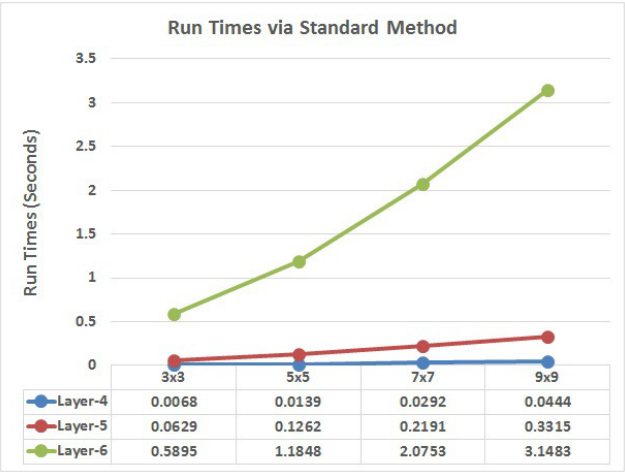


Figure 12: Comparison of edge detection run times for standard method using multi-scale operators for three image sizes.

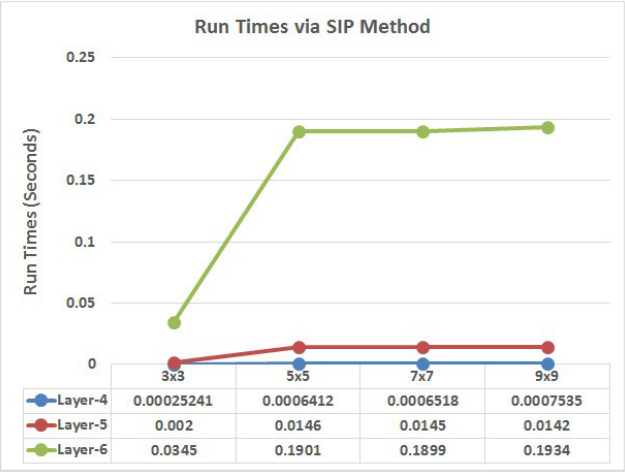


Figure 13: Comparison of edge detection run times for SIP based method using multiscale operators for three image sizes.

4.3. SIP Corner Detection at Mutltiscale Levels

We applied the SIP multiscale approach to image corner detection, in which we used the Harris corner detector to generate the SIP operators. The results of

corner detection based on operators with size of 5×5, 7×7 and 9×9 are given in Fig. 14, based on traditional Harris corner detection and the SIP method. Again, it can be seen that the SIP-based approach to operator implementation can be used successfully at multiple scales.

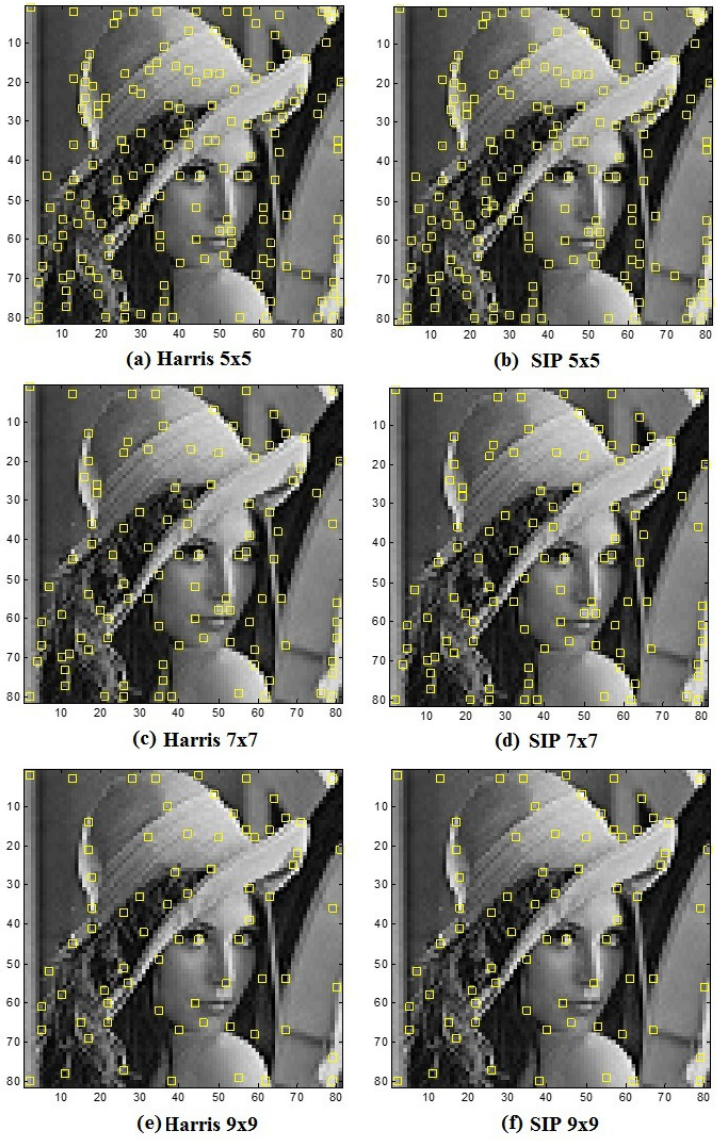
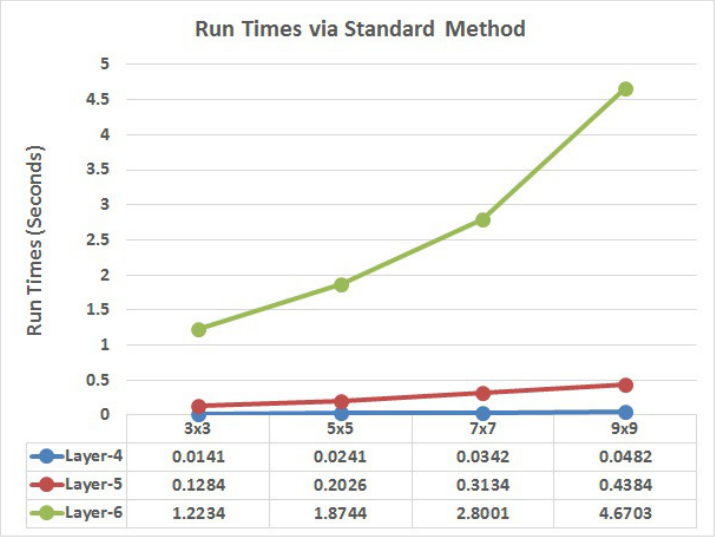


Figure 14: The results of corner detection from SIP multiscale operators.

The run-times for standard and SIP-based corner detection are provided in Fig. 15 and Fig. 16, respectively. Recall that the Harris corner detector is based on calculation of Gaussian smoothed image derivatives,

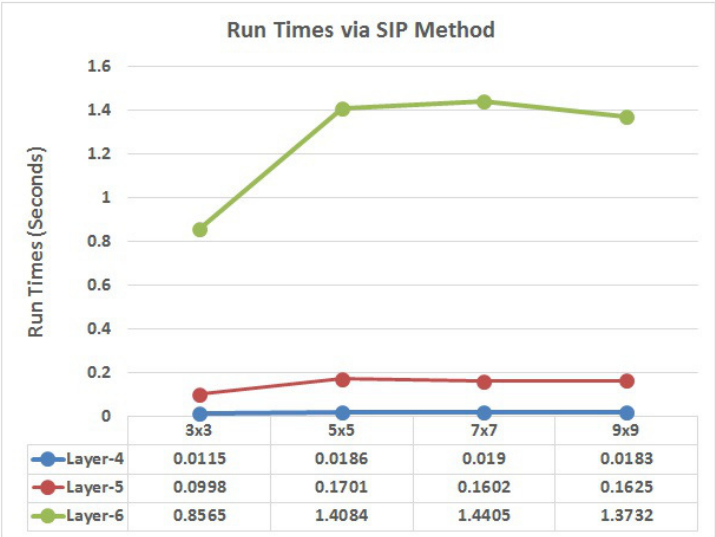
$$\mathbf{H}_{corner} = \mathbf{G}_{\sigma} \begin{pmatrix} \mathbf{I}_x^2 & \mathbf{I}_{xy} \\ \mathbf{I}_{xy} & \mathbf{I}_y^2 \end{pmatrix} \tag{12}$$

where $\mathbf{G}_{\sigma}$ is a Gaussian smoothing function, and $\mathbf{I}_x$ and $\mathbf{I}_y$ are the partial derivatives of image $\mathbf{I}$. Therefore, unlike edge detection that involves a single convolution, Harris corner detection involves five convolutions, with two convolutions for the image derivatives $\mathbf{I}_x$ and $\mathbf{I}_y$ and three for Gaussian smoothing as in Eq. (12). (As mentioned above, only the run time for SIP convolution is considered and the times taken for SIP conversion and generation of eye tremor images are not included). Because the SIP-based convolution can be performed in matrix-vector multiplication and there is no need to convert the outcomes of each convolution to square images, it is more efficient than standard 2D image processing. As can be seen from Fig. 15, for the standard method, the time increases as the sizes of the operators and images increase. As expected, for the SIP-based method, Fig. 16, the time increases from 3×3 to 5×5, and then it remains similar for 5×5, 7×7 and 9×9, similar to the observation shown for edge detection.



	3x3	5x5	7x7	9x9
Layer-4	0.0141	0.0241	0.0342	0.0482
Layer-5	0.1284	0.2026	0.3134	0.4384
Layer-6	1.2234	1.8744	2.8001	4.6703

Figure 15: Comparison of corner detection run times for standard method using multi-scale operators for three image sizes.



	3x3	5x5	7x7	9x9
Layer-4	0.0115	0.0186	0.019	0.0183
Layer-5	0.0998	0.1701	0.1602	0.1625
Layer-6	0.8565	1.4084	1.4405	1.3732

Figure 16: Comparison of corner detection run times for SIP based method using multiscale operators for three image sizes.

5. Discussion

5.1. Related Work via HIP Framework

A spiral addressing scheme has been developed previously for images represented on a rectangular lattice in [20], but based on an addressing scheme using base five arithmetic, rather than base nine, as in our case. The Spiral Honeycomb Image Algebra (SHIA) was originally developed in [2] to perform primitive image transformations on a hexagonal lattice and applied for translation, rotation, and scaling. The SHIA algorithm is related to the hexagonal Image Processing (HIP) framework introduced in [3]. To tackle the issue of lack of hexagonal image capture devices and technologies, SHIA was extended to adapt its mathematical structure to incorporate a rectangular lattice, in which Fourier transforms are computed on signals that are sampled on a rectangular lattice. Because the focus of our paper is the addressing scheme, we discuss the difference between our SIP addressing scheme and the SHIA addressing approach here.

To facilitate SHIA on a rectangular lattice, the extension of SHIA [20] used a base five spiral address. Recall for the hexagonal lattice, spiral counting is performed in base seven with sixty-degree rotations [3] as we move from the centre of each cluster or neighbourhood of pixels to the next, at each level in the pixel indexing scheme. For the base five case used in SHIA, spiral counting is adapted naturally from that used on a hexagonal lattice by changing the base from seven to five, and correspondingly the 60 degree rotations used in HIP to 90 degree rotations in SHIA. Fig. 17 (a) gives an example originally presented in [20], on which we have superimposed the region containing nine pixels (a layer-1 neighbourhood with size 3×3), as used in our SIP addressing scheme, and shown enlarged in Fig. 17 (b) for comparison.

It can be seen that the signals from five pixels are sampled in SHIA, whereas nine pixels are sampled in SIP, and in both cases directly from the rectangular lattice. In the case of SHIA, such addressing results in a fractal-like image representation due to partial representation of the image within each neighbourhood

(as demonstrated in [20]). As SIP uses a base nine addressing scheme as shown in Fig. 17(b), the neighbourhoods or "clusters" of pixels on which the indexing scheme is built are the more familiar compact 3×3 regions on which image processing operators are commonly developed. So within such regions, SIP uses complete sampling and only shifts the order of sampling, rather than carrying out partial sampling as in the SHIA scheme.

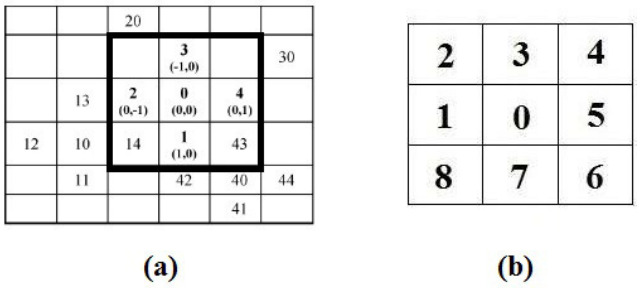


Figure 17: Comparison of addressing scheme for 3×3 via: (a) base five addressing via SHIA [20] and (b) base nine addressing by SIP. It shows that SIP fully represents the nine pixels rather than partially sampling the neighbourhood.

It should be noted that in both SHIA and SIP approaches, there is a one-to-one mapping of the pixel values in the original 2D rectangular image array to the pixel values in the 1D indexing array. In SIP we arbitrarily start the direction of indexing to the left and spiral clockwise, whereas SHIA starts the index sequence in a downward direction and then spirals clockwise. Both techniques are essentially invariant to the starting direction of the pixel indexing sequence, as this affects only the order of the pixels, and any corresponding filters are designed to take this into account. Hence in essence the computations for image processing would not be changed by rotations of the initial indexing direction (in the case of SHIA by multiples of 90 degrees, or in the case of SIP by multiples of 45 degrees). The SHIA indexing scheme results in a fractal-like image representation that builds on irregularly shaped neighbourhoods in the image, extending out recursively to larger regions. So large areas of the image around the border have to be neglected. This is similar to what we see when we

use hexagonal imaging: the spiral architecture used for HIP indexing naturally fits a shape that is built recursively from hexagonal neighbourhoods of increasing size as we spiral out from the centre of the image. SHIA is similar, except that the increasingly larger regions interlock more intricately to tessellate the image region, as the shapes of the neighbourhoods are not a "natural fit" to a rectangular lattice in the way that the hexagonal neighbourhoods are the "natural fit" on a hexagonal lattice. The neighbourhoods used in SHIA fit together more like fractals to generate a tessellation of the rectangular lattice, apart from the border regions. For the base 9 approach used in SIP, the neighbourhoods (of increasing size as we spiral recursively out from the centre of the image) are square, and are a "natural fit" to a square lattice, in the same way that the hexagonal neighbourhoods in HIP are a natural fit to a hexagonal lattice). Hence, depending on the original image size and shape, the SIP approach can cover most (or all) of the image without necessarily neglecting border regions, in a way that the SHIA indexing approach cannot.

5.2. Computation Complexity

Computational complexity measures the processing time and computer memory required by the algorithm to solve problems of a particular problem size. To analyse algorithms independently of specific implementations, hardware, or data, an algorithms execution time is related to the number of operations it executes, denoted by big **O**, the order of magnitude.

For discrete, two-dimensional variables **A** and **B**, the convolution of A and **B** is defined as:

$$C(x,y) = \sum_i \sum_j \mathbf{A}(i,j)\mathbf{B}(x-i+1, y-j+1) \tag{13}$$

The algorithmic implementations for standard convolution and SIP convolution are shown in Fig. 18 (a) and (b), respectively. It can be seen that the algorithm for standard convolution has four *for* loops, and hence $\boldsymbol{O}(n^4)$

operations. The SIP convolution can be implemented as matrix-vector multiplication, which results in $\boldsymbol{O}(n^2)$ operations, and therefore it is more efficient than standard 2D convolution.

```
{Implements Standard Convolution (Matrix Multiply): C = C + A convolve B}
for x = 1 : n
    for y = 1 : n
        for i = 1 : n
            for j = 1 : n
                C(x,y) = C(x,y) + A(i+x-1, j+y-1) * B(i,j)
```

(a)

```
{Implements SIP Convolution (Matrix-Vector multiply): c = c + A*b}
for i = 1:n
        for j = 1:n
                c(i) = c(i) + A(i,j)*b(j)
```

(b)

Figure 18: Computation implements for: (a) standard convolution and (b) SIP convolution.

5.3. Comparison of Run Times

The objective of this work was to provide a proof of concept of the SIP architecture. The algorithms presented in this work were all implemented using Matlab, and so we used a Matlab building function *conv2.m* for 2D convolution to compare the performance of SIP based approach.

For fair comparison, all evaluation experiments were carried based on the same computer, with a 64-bit, Windows Intel processor core i7. Faster performance would be obtained using other programming languages, and this is under investigation for future work. Implementation of the SIP framework based on C and parallel implementation are currently under development, and the results will be reported in the future.

As mentioned in the experimental setup, the run time was counted for the convolution only. The SIP conversion and eye tremor construction were not included, as these are considered to be implemented via hardware in the ideal case. An address look-up table was stored for SIP conversion only, but this has no impact on the run time.

6. Conclusion

In this paper, we present a novel "Squiral" (square-spiral) image processing framework that develops a spiral addressing scheme for standard square images. In the SIP framework, images can be converted easily from the existing typically rectangular coordinate system. Therefore, SIP-based approaches can be incorporated directly with existing hardware systems and state-of-the-art image processing methods for square images. The SIP-based approach enables fast image processing by introducing a non-overlapping convolution in conjunction with an eye tremor model. The applications of the SIP-based approach to image feature extraction (edge and corner detection) demonstrate the efficiency of the proposed approach. The extension of the SIP approach to multiple scales further suggests the potential of SIP in a wide range of applications. Future work will consider the development of the SIP-based interest point detector and SIP feature representation with applications for real-time image and video retrieval.

Acknowledgement

The research leading to these results has received funding from the European Communitys Seventh Framework Programme under grant agreement No. 607691, SLANDAIL (Security System for Language and Image Analysis). The materials presented and views expressed here are the responsibility of the author(s) only. The EU Commission takes no responsibility for any use made of the information set out.

References

References

[1] I. Her, Geometric transformations on the hexagonal grid, IEEE Transactions on Image Processing 4 (1995) 1213–1222.

[2] P. Sheridan, T. Hintz, D. Alexander, Pseudo-invariant image transformations on a hexagonal lattice, Image and Vision Computing 18 (2000) 907–917.

[3] L. Middleton, J. Sivaswamy, Hexagonal image processing: A practical approach, Springer Science & Business Media, 2006.

[4] A. Fayas, H. Nisar, A. Sultan, Study on hexagonal grid in image processing, in: The 4th International Conference on Digital Image Processing, 2012, pp. 7–8.

[5] X. He, W. Jia, J. Li, Q. Wu, T. Hintz, An approach to edge detection on a virtual hexagonal structure, in: Digital Image Computing Techniques and Applications, 2007, 9th Biennial Conference of the Australian Pattern Recognition Society on, IEEE, pp. 340–345.

[6] S. J. Liu, S. Coleman, D. Kerr, B. Scotney, B. Gardiner, Corner detection on hexagonal pixel based images, in: Image Processing (ICIP), 2011 18th IEEE International Conference on, IEEE, pp. 1025–1028.

[7] S. A. Coleman, B. W. Scotney, B. Gardiner, A biologically inspired approach for fast image processing., in: Machine Vision Applications (MVA), 2013 12th IAPR International Conference on, pp. 129–132.

[8] B. Scotney, S. Coleman, B. Gardiner, Biologically motivated feature extraction using the spiral architecture, in: Image Processing (ICIP), 2011 18th IEEE International Conference on, IEEE, pp. 221–224.

[9] P. Sheridan, Spiral Architecture for machine vision, Ph.D. thesis, 1996.

[10] O. S. Packer, D. M. Dacey, Receptive field structure of h1 horizontal cells in macaque monkey retina, Journal of Vision 2 (2002) 1–10.

[11] A. Róka, Á. Csapó, B. Reskó, P. Baranyi, Edge detection model based on involuntary eye movements of the eye-retina system, Acta Polytechnica Hungarica 4 (2007) 31–46.

[12] H. von Helmholtz, J. P. C. Southall, Treatise on physiological optics, volume 3, Courier Corporation, 2005.

[13] S. Martinez-Conde, S. L. Macknik, D. H. Hubel, The role of fixational eye movements in visual perception, Nature Reviews Neuroscience 5 (2004) 229–240.

[14] Y.-P. Wang, Image representations using multiscale differential operators, IEEE Transactions on Image Processing 8 (1999) 1757–1771.

[15] T. Lindeberg, Feature detection with automatic scale selection, International journal of computer vision 30 (1998) 79–116.

[16] S. Mallat, S. Zhong, Characterization of signals from multiscale edges, IEEE Transactions on pattern analysis and machine intelligence 14 (1992) 710–732.

[17] Z. Farbman, R. Fattal, D. Lischinski, R. Szeliski, Edge-preserving decompositions for multi-scale tone and detail manipulation, in: ACM Transactions on Graphics (TOG), 2008, volume 27, ACM, p. 67.

[18] Y.-P. Wang, Q. Wu, K. R. Castleman, Z. Xiong, Chromosome image enhancement using multiscale differential operators, IEEE Transactions on Medical Imaging 22 (2003) 685–693.

[19] D. Kerr, S. Coleman, B. Scotney, Efficiently scaling edge detectors (2011).

[20] P. Sheridan, A method to perform a fast fourier transform with primitive image transformations, IEEE transactions on image processing 16 (2007) 1355–1369.